云原生边缘计算：探索与展望

曾德泽[1]，陈律昊[1]，顾琳[2]，李跃鹏[1]
（1. 中国地质大学（武汉）计算机学院，湖北 武汉 430074；2. 华中科技大学计算机科学与技术学院，湖北 武汉 430074）

摘　要：云原生计算基于低开销容器化的运行方式非常契合边缘计算，因此，提出将云原生技术应用于边缘计算，发挥云原生的优势，使资源管控对应用开发部署透明化。考虑相较于云计算，边缘计算具有资源广分布、高异构、多碎片特征，亟须算网协同的资源管控。根据云原生相关技术的发展现状，整合软件定义网络与网络功能虚拟化等未来网络技术，提出全栈式云原生边缘计算架构。在此基础上，考虑容器的层次化特性，提出适用于有限边缘计算资源的低开销容器部署方法，分析探讨云原生边缘计算所面临的挑战。

关键词：云原生；边缘计算；容器；微服务
中图分类号：TP393
文献标识码：A
doi: 10.11959/j.issn.2096−3750.2021.00206

Cloud native based edge computing: vision and challenges

ZENG Deze[1], CHEN Lvhao[1], GU Lin[2], LI Yuepeng[1]
1. School of Computer Science, China University of Geosciences, Wuhan 430074, China
2. School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan 430074, China

Abstract: Cloud native computing, featured by low-cost container technology, well fits edge computing. It was proposed to apply cloud native computing to make edge computing resource management and control transparent to application development and operation. Compared with cloud computing, edge computing resources are widely distributed, highly heterogeneous, and fragmented, which called for collaborative resource management and control. According to the development status of cloud native related technologies, through the integration of future networking technologies such as soft were defined network and network function virtualization, a full-stack cloud native based edge computing architecture was proposed. Then, considering the hierarchical characteristics of containers, a low-overhead container deployment optimization problem for resource-limited edge computing was studied. Finally, the development challenges faced by cloud native based edge computing were discussed.

Key words: cloud native, edge computing, container, micro service

1　引言

为了应对海量终端快速增长的高算力需求，通过计算卸载等方法就可以利用云计算的海量计算资源，这种方法得到了人们的广泛认可。但“端一云”间高时延难以满足诸多实时应用的低时延要求，边缘计算概念应运而生。边缘计算通过将计算需求转移至靠近用户的一侧，利用网络边缘的计算资源承载云计算服务，利用“数据上行、计算下行”的方式，突破了“终端＋数据中心”两级架构的局限性，可以满足应用对时延与带宽的需求[1-3]。边缘计算利用其独特的地理位置优势以低时延、高带宽的方式提供信息技术服务环境和云计算能力，被视为物联网、5G、大数据、人工智能等领域的关键支

收稿日期：2020−11−30；**修回日期：**2021−02−05
通信作者：顾琳，lingu@hust.edu.cn
基金项目：国家自然科学基金资助项目（No.61772480，No.61972171）；之江实验室开放课题（No.2021KE0AB02）
Foundation Items: The National Natural Science Foundation of China (No.61772480, No.61972171), The Open Research Projects of Zhejiang Lab (No.2021KE0AB02)

撑技术之一，共同促进着工业互联网、车载网络、智能驾驶、虚拟现实/增强现实（VR/AR, virtual reality/augmented reality）、智慧医疗等行业产业的变革[4-5]。特别是随着6G被提上日程，利用边缘计算发展边－端融合系统，通过算网融合调度优化来发挥6G的超低时延优势，被视为未来计算的新兴发展趋势[6]。计算优先网络[7]、算力网络[8]、多接入边缘计算[9]等与边缘计算相关概念与技术，说明了边缘计算在未来网络与计算中的重要性，引发了各行各业的广泛关注。

从产业现状看，阻碍边缘计算发展的主要因素之一是边缘服务的开发、管控以及边缘应用生态的构建。尽管边缘计算资源管理与任务调度优化已经得到广泛研究和关注[9-10]，但截至目前，业界尚未有一套统一标准的资源管理与任务管理框架。尽管边缘计算与云计算具有高度的相似性，但由于其资源异构、设备分布广、资源碎片等特点，传统云计算解决方案（如OpenStack）在应用于边缘计算过程中存在诸多不适与局限，包括：1) 复杂性，边缘设备能力差异大、网络接入方式多、类型多样的终端设备和服务需求的不同导致复杂性大；2) 难预测性，终端设备的移动性和随时间变化的任务需求使得难以预测；3) 不确定性，边缘设备的资源异构性和任务负载的动态变化造成服务需求的不确定性；4) 高动态性，云数据中心内在的数据和服务按需下行需求造成高动态性。通过上述分析表明，打造边缘计算平台的“操作系统”势在必行。

尽管存在差异，云计算的发展历程仍为边缘计算提供了参考与借鉴，诸多解决方案已尝试将云计算相关框架与技术迁移至边缘计算，并根据边缘计算的特点进行定制改进，如针对边缘计算资源广分布的特征对OpenStack进行定制化改进[11]。近年来，云原生正逐渐成为云计算的发展主流。以Kubernetes为代表的云原生编排系统，被广泛认为将成为分布式系统的核心操作系统。云原生通过容器化、微服务化、松耦合化服务，实现基于服务的快速按需应用编排构建，满足快速迭代的需求驱动应用开发模式，成为软件开发的主导力量[12]。边缘计算作为云计算的拓展，也可有类似的发展思路。硬件低耦合的轻量级云原生支撑技术（如容器）较契合边缘计算的资源特征，容器（如Docker）能够轻量级地实现多个用户之间的隔离，进而实现资源的弹性伸缩管理以及应用的注册、发现、编排、发布，提升应用开发运行效率。此外，相较于云计算，边缘计算的异构性更高，其资源形态多样、网络接入多样、访问特征多样，由应用开发运行维护人员直接管理物理资源，挑战大、效率低。因此，通过容器抽象资源，让资源管控对应用开发运行维护人员透明，具有十分重要的意义，将云原生应用拓展至边缘计算平台，已经得到了学术界和工业界的认可。现在已有一些针对边缘环境的解决方案，如Kubernetes的简化版K3s和华为技术有限公司于2018年发布的Kubernetes原生边缘计算平台KubeEdge等框架，迅速得到了工业界的广泛认可。

云原生边缘计算通过容器等支撑技术实现了底层物理计算资源的抽象与软化，解耦软件开发运行维护与底层资源管控。事实上，除了计算资源以外，网络资源管控也是边缘计算需要考虑的另一个重要方面。相较于云计算，边缘计算网络异构性更强，表现在接入方式（有线与无线并存）异构、带宽异构、可靠性异构等多个方面。同时，边缘计算具有非规则的网络拓扑。边缘计算是移动互联网的重要支撑，移动互联网的蓬勃发展对边缘计算的资源管控提出了更高的要求。从网络角度看，邬江兴院士指出：互联网的多元化发展，导致当前僵化的网络运行机制下传输控制、资源管理、配置维护等复杂性倍增，网络效率低下，用户体验差[13]。针对该问题，以软件定义网络和网络功能虚拟化为代表的未来网络技术以网络开放为目标，打破传统刚性的网络架构和基线技术对网络多元化发展、个性化服务、创新应用发展的制约。软件定义网络和网络功能虚拟化事实上实现了网络层的软化，通过软化实现开放，进而支撑定制创新，在云计算中已得到了广泛认可与应用。

综合当前边缘计算的发展困境以及云计算的前沿发展趋势，本文认为软化是推动边缘计算发展的潜在思路。本文将云原生技术以及未来网络技术融合，提出了全栈式云原生边缘计算架构，实现算网融合的资源管控。针对边缘资源有限特征，通过发掘利用容器的分层结构特征，研究了低开销的边缘服务部署优化方法。最后，对云原生边缘计算将面临的发展挑战进行展望。

2 云原生计算的发展现状与趋势

2.1 云原生架构：微服务和无服务器计算

云原生计算[14]通过引入灵活的网络资源调度方式，实现动态的资源编排，成为云计算网络开发

和软件部署最快速有效的方式，得到了工业界的广泛认可。云原生计算基金会定义云原生计算的特征属性为：面向微服务、基于容器化封装和自动化管理。在云原生的架构中，应用程序被开发为无状态和松散耦合的微服务，从而提高服务的重用性和可靠性。以云原生的方式对应用进行运行维护具有诸多优势，能够适应 DevOps 模式、持续集成等应用研发需求。如当一个服务实例在运行中失效时，另一个服务实例可以快速生成并立即顶替其功能；当服务请求剧烈变化时，能够灵活地实现服务实例的缩放，提高资源利用率，保障用户服务质量。接下来，简要介绍云原生计算的架构与支撑技术。目前，云原生计算主要有微服务和无服务器计算两种主流架构[15]。

微服务通过将一个大型单体应用分解为多个小型模块进行部署，这种方法的优势是可以构建、测试和部署单个服务，不会对整个产品或其他服务产生不良影响[12]。微服务能够实现更便捷的发布，开发者可以在最初只向子集发布新特性，然后在该特性达到目标期望后转向整个用户群发布新特性。微服务所具有的敏捷性、可靠性和可伸缩性很契合应用开发运行维护的需求。

无服务器计算是将基础设施服务器层抽象出来的概念[16]，这样可以帮助开发人员专注于构建应用，不必考虑硬件服务器的可伸缩性、可用性和安全性。从使用的角度来看，应用程序应尽可能多地使用网络和存储，这意味着当程序没有被使用时，硬件资源不会被闲置；当使用量激增时，基础设施可以在任何程度上扩展，而不必立即手动提供新服务器[12]。

微服务体系结构允许小团队的开发人员专注于单独的服务，每个团队提供一个特定的任务，而无服务器计算则帮助团队以最少的精力启动和运行这些微服务。微服务仍涉及一定量的计算资源集群管理[12,17-18]，而无服务器计算使得应用开发者可以专注于代码编写并定义应该触发代码执行的事件，并将其留给云来处理其余的事。

2.2　云原生支撑技术

云原生的出现需要开发者提出新的协议及架构，一些比较成熟的技术如容器[12]、微服务调度工具（如 Kubernetes）、服务网格等，为云原生的实现提供了巨大的技术支持与经验。云原生的关键支撑技术如下。

1) 容器

容器的本质是一个进程，通过对该进程进行隔离和资源控制，其在运行时不会相互干扰。同时，容器具有良好的移植性，可以在不同的操作系统中良好运行。以前，开发者通常使用虚拟机完成某些功能，但是虚拟机的系统开销较大，并且不利于程序的移植与部署。与虚拟机相比，容器更加轻量化，打包下载都更方便，其中最著名的容器是 Docker。Docker 是一个开源容器引擎，可以让开发者打包其应用以及依赖包到一个轻量级、可移植的容器中，然后发布到任何流行的操作系统中。在运行时，Docker 的性能开销几乎可以忽略不计。Docker 占用资源少、部署快，每个应用可以被打包成一个 Docker 镜像。Docker 容器中每个应用不需要与其他应用组合，也不依赖于生产环境基础结构，能在研发、测试、生产过程中为应用提供一致的环境。Docker 使用客户端—服务器模式，利用远程应用程序接口（API, application programming interface）来管理和创建 Docker 容器。Docker 容器通过 Docker 镜像创建，镜像就像容器的模板，每次创建容器都依赖于已有的镜像。

2) 容器管理器

仅依靠容器并不能满足开发者的需求，现在，一个集群中一般有上万个容器镜像，为了管理这些容器，如控制容器的生存周期，对容器进行迁移，对这些容器间的流量进行调度等，开发者需要一个完备的治理框架，目前 Kubernetes 是应用较广泛的治理框架。Kubernetes 简称 K8s，是一个可移植、可扩展的开源平台，用于管理基于容器的微服务集群。在 Kubernetes 中，可以创建多个 pod（Kubernetes 的基本单元），每个 pod 中可以部署多个容器，每个容器中可以部署一个服务，然后通过内置或自定义的负载均衡策略，实现对一组微服务的管理、注册和访问。开发者可以使用 Kubernetes 的 kubectl 组件对其下的各个服务进行管控。

3) 服务网格

尽管在很多情况下 Kubernetes 能够完成微服务治理功能，但开发者仍然会遇到其他问题。在实际生产环境中，由于微服务的实现方式不同，如果要使微服务之间进行通信，开发者需要预先协调通信接口和方式。此外，流量的管控与调度由于要考虑环境中的各种因素，所以开发过程并不简单。因此，服务网格应运而生[19]。Kubernetes 与服务网格对比

如图 1 所示，作为服务之间通信的基础设施层，服务网格可以分离微服务中的通用功能，如服务注册发现、负载均衡、熔断降级、流量管理、监控等功能，极大地弥补 Kubernetes 使用时的一些不足。

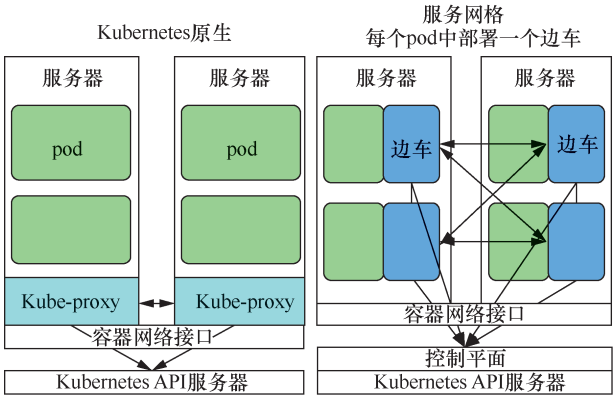

图 1　Kubernetes 与服务网格对比

在服务网格中，每个微服务通过边车代理其服务间通信，因而大量的微服务与对应的边车会表现出网格的形式，这就是服务网格名称的由来。服务网格可以在多种环境中部署，最常用的是 Kubernetes。服务网格（如 Istio 框架）与 Kubernetes 有较好的兼容性，Kubernetes 对应用进行生命周期管理，服务网格提供应用间的流量、安全管理以及可观察性。

云原生支撑技术框架对比如图 2 所示，Kubernetes 的 Kube-proxy 组件实现了流量在 Kubernetes 服务中多个 pod 实例间的负载均衡，但是 Kube-proxy 的设置是全局的，无法对单独的 pod 进行精细的管理。而服务网格将 Kubernetes 中的这一功能分离出来，部署在边车中，不再需要 Kubernetes 的 Kube-proxy 组件支持，通过更接近微服务应用层的部署来管理服务之间的通信、流量管理、负载均衡和可观察性。总体来讲，Kubernetes 使用 Docker 容器技术来部署和管理微服务，实现微服务的自动部署、自动重启、自动复制、自动扩展等功能，服务网格则可以部署于 Kubernetes 上，负责提供应用间更灵活的流量控制、安全管理以及可观察性。

4) Unikernel

Unikernel 是一个特殊的、单地址空间的机器镜像，它实现了底层硬件资源的直接取用，免去了不必要的硬件抽象。Docker 是一种有效的应用管理容器技术，它解决了应用的可移植性问题。相比于虚拟机，Docker 的镜像已经小了很多，但还是有几百兆。Unikernel 则进一步减小了自身的体积，其体积只需要几十兆甚至几兆。Unikernel 在构架中去除了操作系统，其应用直接运行在 Hypervisor 或者硬件上，由于不包含许多应用不需要的包和依赖，从而有效改善了资源的利用情况。Unikernel 具有以下优势：①体积小，Unikernel 只需要包含应用必须的依赖和包，大大减小了自身的体积；②速度快，Unikernel 中没有其他不必要的程序，有效减少了多进程之间的任务切换和启动，使得中央处理器（CPU, central processing unit）能够高效运行，启动速度也非常快，通常只有 20 ms 甚至几毫秒，使它在用户需要时进行启动响应。Unikernel 也由此被视为实现云原生的重要支撑技术之一。

2.3　云原生边缘计算

云原生助力边缘计算，已经得到了工业界的认可，出现了一些应用较广泛的框架，如阿里云近期开源的项目 Openyurt。Openyurt 是基于原生 Kubernetes 构建的框架，用户可以使用 Openyurt 在边缘环境中管理运行的微服务。它克服了一些边缘场景的限制，例如，如何最小化设备和工作负载之间的长距离网络流量，如何解决边缘场景中的可靠性，如何进行安全验证，如何降低传输时延等。Openyurt 提供了完全的 Kubernetes API 兼容性，支持 Kubernetes 的所有特性，它也提供了一种工具可以让本机的 Kubernetes 转化为边缘状态，还能提高集群在边缘场景中的稳定性。

KubeEdge 是云原生在边缘计算中拓展的典型

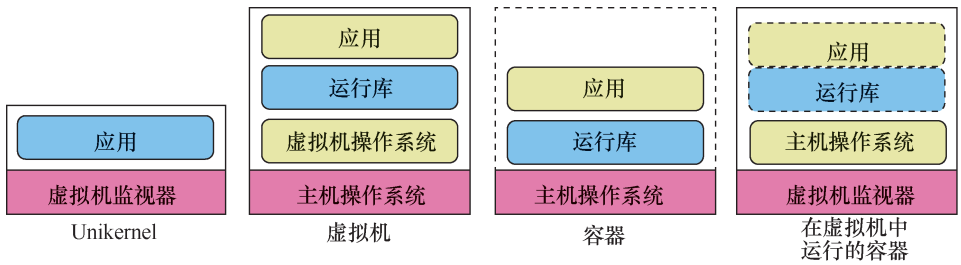

图 2　云原生支撑技术框架对比

架构，其基于 Kubernetes 架构提供了许多边缘场景的功能支持，如离线运行能力、边云协同能力等。该架构分为云端组件和边缘组件。在云端组件中，用户通过 Kubectl 命令行对目标对象的预期状态发出命令，由 Kubernetes 的 API 服务器接收并使用调度器调度对象；在边缘组件中，所有组件设计的原则为“简单至上”，降低了边缘组件的资源占用、发生故障的概率以及维护的难度。

K3s 是另一种经典的被运用在边缘计算中的架构，传统的 Kubernetes 的组件很复杂，导致其在边缘设备性能不足的背景下工作较困难，而 K3s 仅作为单一的二进制文件进行打包和部署，安装迅速便捷。同时，K3s 删除了很多对于运行最低限度的集群来说不重要的组件，加入了一些新的必要元素，使得其可以适用于边缘计算场景。

尽管现在已经有一些架构可以帮助开发者应对边缘场景，但这些架构仍然存在一些问题，它们大多只考虑了边缘环境的一部分，如只考虑了服务治理，而忽略了网络协议等也是可以融合的。因此，本文将进一步地将服务网格、软件定义网络、网络功能虚拟化等技术融合，提出全栈式架构。

3　全栈式云原生边缘计算架构

云原生计算变革了应用开发与运维模式，通过底层物理计算资源的抽象与软化，应用开发运行维护人员集中关注应用自身，不需要考虑底层物理资源。同时，针对传统网络架构刚性基线技术对网络灵活管控的局限、对网络资源与计算资源协同调度的制约，以软件定义网络与网络功能虚拟化为代表的未来网络技术提供了新的解决途径。融合云原生计算与未来网络技术，本文提出了面向边缘计算的全栈式云原生计算架构，实现边缘网络资源、计算资源的协同管理，全栈式云原生边缘计算架构如图 3 所示。当前分布式计算系统结构的发展趋势是数据层与控制层的分离，如软件定义网络通过解耦并集中控制层面，极大地提升了计算机网络的灵活性和可定制性；服务网格提供控制器管理边车所代理的服务的流量。因此，本文认为，解耦数据层与控制层是未来分布式系统架构的发展趋势之一。图 3 中所提出架构的最上层为控制层，包含面向多个不同功能组件的控制器，控制层的各个控制器组件管控其下各层的不同组件。

1) 物理层

物理层是整个系统的最底层，包含各种边缘计算物理资源，如计算机、网络设备、通信设备、传感设备、存储设备等。从资源类型来看，边缘计算与支撑云计算的数据中心类似。但从资源特征来看，边缘计算与云计算有很大区别，主要表现在资源高异构与广分布两个方面。云计算数据中心一般由单一的主体运行，其物理设备相对比较统一。由于边缘计算接入门槛低，各类主体均可提供基础物

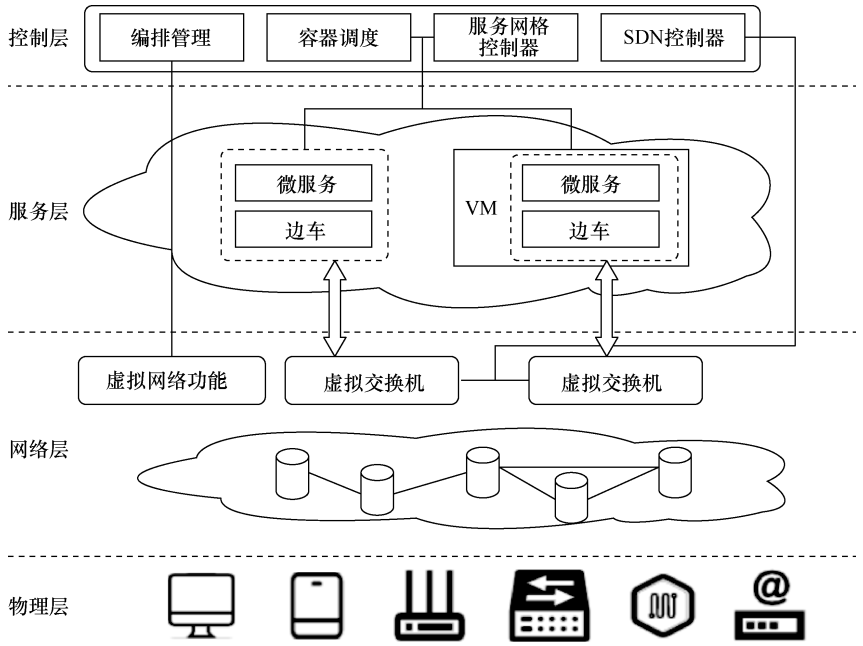

图 3　全栈式云原生边缘计算架构

理设施接入边缘计算平台。如移动运营商可通过靠近蜂窝网通信基础设施的计算设备接入边缘计算提供物理资源，城市中的小区也可建设小型数据中心。各基础设施提供商的物理资源难以统一，不仅表现在资源容量方面，同时表现在资源架构方面。如计算机可能为x86架构，也可能为ARM（advanced RISC machine）架构，且同时可能部署不同的硬件加速器，如图形处理器（GPU, graphics processing unit）、现场可编程逻辑门阵列（FPGA, field programmable gate array）等。此外，由于边缘计算靠近终端用户，其网络接入方式多样，不仅包括有线网络接入，还包括各类无线网络接入（如 4G、5G、Wi-Fi 等）。多元化基础设施的提供，不仅引发了边缘计算的高异构，同时也导致了资源分布广的特征。因此，边缘计算不具备云计算数据中心相对规则的网络拓扑（如 FatTree、Bcube 等），其网络拓扑由于地理分布式的计算资源供给具有非规则属性。因此，物理资源的高异构与广分布对资源管控带来了极大的挑战，阻碍了边缘计算的发展。抽象广分布的异构物理资源，使得边缘计算物理资源特征对应用开发运行维护透明，对边缘计算的发展有重要推动作用。

2) 网络层

由于物理层资源的广分布特征，使得网络成为串联各类异构资源的核心。同时，如前所述，传统封闭刚性的网络技术阻碍了边缘计算应用的开放式发展。因此，在本文提出的全栈式网络架构中，为实现算网协同的资源调度，单独抽象一层独立的网络层。网络层不仅包括交换机、路由器等基础通信物理设施，还包括驱动软件定义网络功能实现的各类基础软件（如 Open vSwitch 等）以及各类虚拟网络功能。软件定义网络通过将控制平面从数据平面中分离出来，使开发者在不改变硬件设备的前提下，可以更灵活地进行网络管理，从而实现对上层应用（或服务）间的开放式流量管理与个性化定制。同时，通过网络功能虚拟化技术为应用间的流量管理提供各类网络服务。通过将传统基于硬件实现的各类虚拟网络功能软化，使网络设备功能不再依赖于专用硬件，实现对各类网络服务的灵活管理、新业务的快速开发和部署。

3) 服务层

服务层直接采用云原生计算架构，其核心是在物理计算资源上抽象出各种服务构建边缘应用。因此，服务层同时涉及容器与虚拟机两种虚拟化技术。首先，与云计算数据中心类似，以虚拟机的形式抽象物理计算资源，提升物理资源利用率。在虚拟机中，可进一步地部署容器用于服务实例的实现与部署。通常情况下，一个容器不能满足一个微服务的所有需求，因此主流的容器管理器（如 Kubernetes）会将一些容器联合起来，将其称为 pod，并将 pod 作为最基础的处理单元，即服务实例。此外，在边缘计算场景中，由于资源限制，并非所有计算设备都适合部署虚拟机进行物理资源共享。因此，pod 及其所包含的各类容器也可以直接部署在物理服务器上。无论 pod 部署在虚拟机上还是物理服务器上，均对应用的开发与运行维护透明。也就是说，边缘应用开发运行维护人员的关注对象为 pod 或由 pod 提供的基础服务，不需要关注物理资源的管控，后者由边缘设施提供商进行运行维护。特别注意的是，在本文提出的全栈式架构中，引入了服务网格实现异构服务间的标准化流量通信管理。如前所述，每个服务将对应一个边车，边车与软件定义网络的数据平面融合，共同构成本文框架中的数据层部分。通过融合，使得网络层的软件定义交换机中的流表能够根据服务的属性信息（如服务名、端口号等）进行定制，如基于 P4 进行网络数据平面编程，而不需要仅按照传统软件定义网络中指定的属性信息进行网络流管控，加强网络管理的纵深与融合程度，从而通过全栈式的方式提供开放、灵活、可定制的网络流管理服务。

4) 控制层

控制层集成了面向各层的管理组件，包括容器编排管理模块、服务网格控制器、软件定义网络控制器和虚拟网络功能编排管理模块 MANO。通过控制层与服务层的剥离，实现全栈式便捷灵活的边缘应用编排管理。容器编排管理模块能够根据应用需求、访问需求、物理资源动态变化，基于设定的规则实现容器全生命周期的管理，包括服务注册、部署、扩容、销毁等。Kubernetes 主节点组件可用于实现这一功能。与之相配合的是服务网格管理器（如 Istio Controller），通过管理各个服务对应边车中的网络流，实现面向服务的负载均衡、安全监测、熔断等运行时操作。软件定义控制器通过南向接口（如 OpenFlow）向软件定义交换机注入流表项，管控数据流在网络层

中的行为。服务网格控制器与软件定义控制器能够相互配合，从不同层面对网络流进行管理。前者主要从服务的角度出发，从逻辑上管控服务间流量；后者是对前者逻辑管控的物理实现。网络功能管理模块与容器管理模块类似，但两者的管理对象不同，网络功能管理模块针对各个虚拟网络功能的生命周期进行管理。控制层的各个模块通过相互配合，能够实现算网融合的高效调度，符合当下分布式计算的发展趋势。

4　面向边缘计算的低开销微服务部署

与资源相对丰富的云计算数据中心不同，边缘计算资源容量匮乏。在应用云原生技术于边缘计算时，资源开销是需要考虑的一个重要方面。微服务部署是实现云原生应用的重要基础，高效地部署微服务对云原生边缘计算具有重要意义。云原生的支撑技术之一——容器也为实现低开销微服务部署提供了新的优化点，尽管边缘计算中的微服务部署优化研究已被广泛关注[20-21]，但现有的研究一般将容器视为轻量级的虚拟机，往往忽略了其分层属性。因此，本节将探讨如何发掘利用容器的分层属性实现低开销的微服务部署优化。

4.1　容器分层与问题陈述

容器在实现时不是一个整体，而是按照分层结构组织。如加入某容器基于 Ubuntu 系统实现，且需要 MySQL 数据库的支持。可以理解为，容器的底层是一个 Ubuntu 镜像，其上叠加了一个 MySQL 层，再在上面叠加所需的操作，这样便能组成一个完整的服务功能。显然，当在同一台物理机上部署多个容器时，如果容器间进行层共享，就能够有效降低开销。一方面，在镜像拉取时，层共享使得不需要拉取冗余的层，不仅降低了网络开销还有利于服务的快速启动；另一方面，边缘服务器也不需要冗余存储容器的层，能够有效降低存储资源开销。

边缘计算环境中的分布式服务器的资源有限，同时需要部署的容器具有不同的计算与存储资源需求，如何综合考虑容器的分层特性，实现边缘计算环境中的低开销容器部署，对云原生边缘计算具有重要意义。首先构建分层敏感的低开销容器部署优化理论模型，为便于读者理解，总结了模型中涉及的数学符号与其定义，数学符号与定义如表 1 所示。

表 1　　数学符号与定义

符号	定义
$\mathcal{N}$	常量集合，服务器集合
$\mathcal{I}$	常量集合，容器集合
$\mathcal{L}$	常量集合，所有容器包含的层集合
P_n	服务器 $n\in\mathcal{N}$ 的计算资源
S_n	服务器 $n\in\mathcal{N}$ 的存储资源
B_n	服务器 $n\in\mathcal{N}$ 的通信资源
C_l	容器中的 $l\in\mathcal{L}$ 层所占的存储资源
z_i^l	二元常量，容器 $i\in\mathcal{I}$ 是否需要层 $l\in\mathcal{L}$
x_n^i	二元变量，容器 $i\in\mathcal{I}$ 是否被部署在服务器 $n\in\mathcal{N}$ 上
y_n^l	二元变量，层 $l\in\mathcal{L}$ 是否被部署在服务器 $n\in\mathcal{N}$ 上
d_i	容器 $i\in\mathcal{I}$ 所需要的计算资源
b_i	容器 $i\in\mathcal{I}$ 所需要的通信资源

4.2　层共享敏感的低开销容器部署优化

本文考虑一个边缘计算环境中包含一个服务器集合 $\mathcal{N}$，其中每台边缘服务器均具有一定的计算资源、存储资源与通信资源。服务器 $n\in\mathcal{N}$ 的计算资源、存储资源与通信资源容量分别表示为 P_n、S_n 和 B_n。拟部署的容器集合为 $\mathcal{I}$，所有容器所包含的层表示为集合 $\mathcal{L}$。不同的层具有不同的大小，即存储空间要求不同。假设层 $l\in\mathcal{L}$ 的大小为 C_l。容器与层的关系可表示为

$$z_i^l=\begin{cases}1, & \text{容器}i\text{需要}l\text{层}\\ 0, & \text{其他}\end{cases} \tag{1}$$

将容器与边缘服务器间的部署关系表示为二元变量，如式(2)所示。

$$x_n^i=\begin{cases}1, & \text{将容器}i\in\mathcal{I}\text{部署在边缘服务器}n\in\mathcal{N}\text{上}\\ 0, & \text{其他}\end{cases} \tag{2}$$

由于容器所需的层已知，容器的部署反映了层与边缘服务器间的部署关系。因此，定义二元变量为

$$y_n^l=\begin{cases}1, & \text{服务器}n\in\mathcal{N}\text{上部署层}l\in\mathcal{L}\\ 0, & \text{其他}\end{cases} \tag{3}$$

显然，如果某个容器 i 部署在边缘服务器 n 上，则意味着 i 所需的所有层均需要部署在服务器 n 上，即存在关系为

$$x_n^i\bullet z_i^l=y_n^l,\forall\, n\in\mathcal{N},i\in\mathcal{I},l\in\mathcal{L} \tag{4}$$

为满足部署的完整性要求，每一个容器均需要部署在一个服务器上，即

$$\sum_{n\in\mathcal{N}} x_n^i = 1, \forall i \in \mathcal{I} \tag{5}$$

考虑边缘服务器的存储容量有限，部署在任何一台服务器上的容器及所需的各层存储资源需求均不能超过服务器能够提供的存储空间。存储空间容量限制可表示为

$$\sum_{l\in\mathcal{L}} y_n^l C_l \leqslant S_n, \forall n \in \mathcal{N} \tag{6}$$

类似地，P_n 为服务器 $n \in \mathcal{N}$ 拥有的计算资源，容器 $i \in \mathcal{I}$ 需要的计算资源为 d_i，则计算资源容量限制可表示为

$$\sum_{i\in\mathcal{I}} x_n^i d_i \leqslant P_n, \forall n \in \mathcal{N} \tag{7}$$

对于通信资源限制，B_n 为边缘服务器 $n \in \mathcal{N}$ 拥有的通信资源，b_i 为服务 $i \in \mathcal{I}$ 的通信资源需求，则有

$$\sum_{i\in\mathcal{I}} x_n^i b_i \leqslant B_n, \forall n \in \mathcal{N} \tag{8}$$

综合考虑以上因素，可将层共享敏感的低开销部署优化问题描述为一个整数线性规化（ILP, integer linear programming）问题，如式(9)所示。

$$\min \sum_{n\in\mathcal{N}} \sum_{l\in\mathcal{L}} y_n^l C_l$$
$$\text{s.t. 式(4)}\sim\text{式(8)} \tag{9}$$

ILP 问题一般被视为 NP-hard 问题，直接求解获得最优解具有指数级计算复杂度，可以通过求解器 Gurobi 获得最优解，但当问题空间大（如边缘计算机数量大、拟部署的容器数量多）时，难以快速获得最优解。因此，本文进一步提出基于所构建 ILP 优化模型的贪心算法求最优解。

式(9)所示优化理论模型由于存在整数项使得计算复杂度高，即二元变量 x 和 y，通过线性松弛将其表示为 0～1 的实数，可将原问题转化为线性规划（LP, linear programming）问题。LP 问题可在多项式时间内获得最优解，但由于线性松弛，使得 x 和 y 所表示的物理含义不能直接反映拟获得的部署映射关系。因此，需要进一步将其还原为 0 或者 1，本文采取贪心算法进行还原。

松弛算法如算法 1 所示，首先对原来的问题进行松弛操作，将 x 和 y 视为 0～1 的实数变量，此时该问题转化为一个 LP 问题，可以通过求解器获得 x 和 y 的实数解。然后，将实数解转化为 0 或者 1，用于表示一个容器放置在哪个服务器上。此处，采用贪心思想对每个服务器计算权重，该权重由两部分组成，一部分是解 LP 问题的结果，另一部分由当前服务器的资源决定，资源越大，所占权重越多。在算法 1 中，根据权重的大小对连续变量进行离散化操作，权重越大，越可能被赋值为 1。当所有容器被放置在服务器上时，这些服务所需要的层便被唯一确定，可算出要在该服务器上部署哪些层。需要注意的是，由于将连续变量离散化后不一定能满足约束限制，所以需要不断地循环这一操作，直到满足所有约束为止。最后，得到所要的结果。

算法 1 松弛算法

输入 服务的集合 $\mathcal{I}$，服务器的集合 $\mathcal{N}$，层的种类集合 $\mathcal{L}$ 等各项参数

输出 部署所有服务的资源开销 result

将离散变量 x_n^i 和 y_n^l 转化为[0,1]的连续变量 X_n^i 和 Y_n^l

此时该问题转化为 LP 问题，可求得 X_n^i 和 Y_n^l

while x_n^i 和 y_n^l 不满足约束条件

for all $i \in \mathcal{I}$ do

for all $n \in \mathcal{N}$ do

每个节点的权重 $w_j = X_j^i (P_j / \sum_{i\subseteq N} P_i + S_j / \sum_{i\subseteq N} S_i + B_j / \sum_{i\subseteq N} B_i)$

按 w_j 的从大到小对 x_n^i 进行赋值，x_j^i 有 $w_j / \sum_{j\subseteq N} w_j$ 的概率为 1，当 x_j^i 取 1 时，其他节点上都不会放置微服务 i

end for

end for

已知服务器 n 上的微服务放置情况 x_n^i，可得 n 上的层放置情况 y_n^l

end while

通过式(9)计算模型所有层占用的存储空间大小 result

输出 result

为了验证本文所提算法的有效性，进行了基于数值分析的模拟实验。随机生成 10 个服务器，其计算资源 P_n 在[10 000, 12 000]内随机取值，存储资源 S_n 在[2 000, 3 000]内随机取值，通信资源 B_n 在[2 000, 3 000]内随机取值。所需部署的容器为 10 个，

层大小在[200, 300]内取值，容器的计算资源需求在[1 000, 2 500]内随机取值，通信资源需求在[600,1 000]内随机取值。为了凸显验证本文所提出层共享敏感的容器部署方案的优势，将传统不考虑层共享的最小开销部署方案与本文方案进行对比。对于层共享敏感的方案，同时给出了通过求解 ILP 模型获得的最优解与基于本文提出的线性松弛算法获得的次优解。容器数量对部署开销的影响如图 4 所示，边缘服务器资源量对部署开销的影响如图 5 所示。

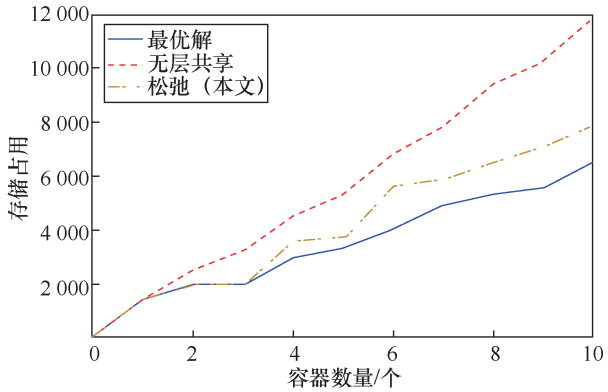

图 4　容器数量对部署开销的影响

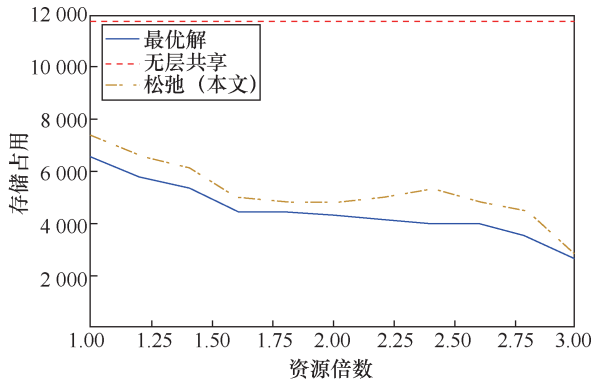

图 5　边缘服务器资源量对部署开销的影响

图 4 主要考虑容器数量的增多对部署开销（以存储占用来衡量）的影响。从图 4 中可以看出，随着容器数量的增加，3 个算法所需要的总存储空间也逐渐增加。对比 3 个算法可以看出，考虑层共享的部署方案在存储资源占用上均优于无层共享的方案。此外，本文所提的松弛算法可以取得逼近最优解的结果。如图 4 所示，当容器数量较少时，能够共享的层数量有限导致规划空间有限，松弛算法可以得到与最优解几乎相同的结果。随着容器数量的增加，占用的空间也越来越多，因此总消耗呈上升趋势。但是由图 4 可以看出，采用层共享方法存储的方式可以明显降低开销，容器数量越多，优势越明显。这是因为，随着容器数量的增加，可共享的层越来越多，层共享的优势也越来越明显。

图 5 展示了服务器资源的提升对部署开销的影响，将资源大小随倍数增加，随着存储空间、计算资源和通信资源的增加，同一个服务器上可部署的容器数量也会逐渐增加，越来越多的容器可以被部署在同一个服务器上，潜在地能够共享更多的层，从而降低容器的总部署开销，而无层共享的方案则一直需要相同的部署开销。实验结果证明了层共享在部署开销方面的优势，当存储空间、计算资源和通信资源增加到一定程度时，所有容器可以部署在一个或少数几个服务器上，此时松弛算法和最优解取得相同结果。

5　展望与挑战

尽管云原生对推动边缘计算发展具有极大的潜力，工业界也相继推出了相关解决方案，但该方向的研究仍处于初始阶段。由于边缘计算环境与云计算数据中心的本质区别，云原生边缘计算还存在诸多挑战。本节将展望并分析可能存在的挑战。

1) 广分布高异构资源管理

边缘计算资源分散在不同地理分布的地方，如用户终端设备、通信基站、智能路由器、边缘服务器等。而云计算中的资源都是集中式的管理，因此云计算的资源管理方式和框架（如 OpenStack）不适用于管理边缘计算分散的资源，亟须发展新的适应广分布高异构边缘资源管理的解决方案。目前，关于边缘计算资源的管控研究大多集中在算法层面，容易忽略在具体部署实践中的某些细节。此外，CPU 架构差异（如 x86 与 ARM）、硬件加速器并存（如 FPGA、GPU 等）、有线与无线共在等特性，均对边缘资源管理带来了极大的挑战。尽管云原生使应用开发运行维护人员能够远离这些细节，但对基础设施管理人员提出了要求，需要其能够设计出上层应用透明，但又对上层应用特征敏感（如本文所述容器的分层结构）的高效解决方案。

2) 分布式服务状态管理

云原生计算中的微服务或者函数功能自身均不保留状态，但实际上并非所有服务或函数都没有状态，因此状态管理成为云原生必须讨论的一个问题。在云原生边缘计算中，这一问题变得更突出，主要由于前述边缘计算平台环境自身的特征。此

外，边缘计算中的服务请求可能具有高度移动性，以及由此导致的时空异构性，这都对云原生边缘计算的状态管理带来了新的挑战。显然，集中式的状态管理不适用于云原生边缘计算，亟须发展分布式服务状态管理方法。由于边缘计算环境的高时空异构动态性，服务会高速动态部署、销毁、扩容，相应的分布式服务状态的存储位置（一份或多份备份）、存储方式（如内存或硬盘）、同步方式（若存在多份状态备份）、更新方式（增量更新或全覆盖更新）等均需要与服务的生命周期管理耦合匹配，实现高效的服务状态管控。

3) 边缘智能的应用

边缘智能随着边缘计算的发展逐渐成为一个热点话题。尽管边缘智能主要指在边缘计算平台上部署的各类人工智能应用，本文认为边缘智能的一个重要应用对象就是边缘计算平台自身。云计算数据中心需要自动化、智能化运行维护方案，广分布高异构且资源供需动态性高的边缘计算平台更需要智能化运行维护。然而，相较于云数据中心，基于边缘智能的边缘平台运行维护面临着环境复杂和算力有限两方面挑战。如海量的微服务需要在拓扑非规则、资源高异构的边缘平台上进行部署、扩容、缩容等操作，同时伴随着服务请求的高度时空动态性，需观测的数据不仅数量大而且维度大，并且对数据处理的时效性要求高。这都与算力有限的边缘服务器构成矛盾，通过云边端融合构建分布式的边缘智能环境，通过智能体协作共同完成云原生边缘计算运行时的智能管控是潜在的发展途径。强化学习（包括深度强化学习）、联邦学习、迁移学习等，这些都是潜在的支撑技术。

4) 云原生边缘应用安全与隐私

云原生的核心支撑技术是容器。相较于虚拟机，容器的不足之处是其低隔离性。低隔离性导致部署在同一物理机或虚拟机上的不同容器不仅存在资源竞争，并且存在安全与隐私风险。然而，云原生难以摒弃低隔离性的容器在于其轻便性，两者很难兼得。因此，在可能有多边缘计算设施提供商以及多边缘服务提供商共存的云原生边缘计算环境中，保障边缘应用的安全与隐私势在必行。本文认为，利用可信执行环境（TEE, trusted execution environment）是潜在的解决方案之一。当前各个主流的 CPU 厂商均推出了自己的 TEE 产品，如 Intel 公司的 SGX、ARM 公司的 TrustZone 等。然而，TEE 为了保障业务的安全保密执行，也引来了诸多必须考虑的因素。如 SGX 的安全内存空间有限，而 TrustZone 的 CPU 要求独占。容器（包括安全容器与非安全容器）的编排调度，如何与 TEE 的特征相适应，是亟待研究的一个方向。

6 结束语

云原生技术在云计算中的应用为边缘计算的发展指明了新的潜在发展方向。本文首先简要介绍了云原生的相关概念与关键技术，在此基础上，针对边缘计算的场景特征，融入软件定义网络与网络功能虚拟化，提出了全栈式的云原生边缘计算架构。该架构通过控制层面与服务层面的抽象剥离，能够实现高效的算网协同边缘资源管理与应用运行维护。进一步地，针对云原生的技术特征，本文研究了容器分层特性敏感的低开销容器部署优化方案，通过该实例研究表明，云原生边缘计算中的资源管控必须考虑云原生自身所引入的新特征。最后，展望了云原生边缘计算发展所面临的新挑战。总之，云原生非常契合边缘计算，尽管它是一个新兴的概念与技术，随着研究的不断深入，未来一定能够得到巨大的发展，期待这个领域内部的技术突破。

参考文献：

[1] XU J, CHEN L X, ZHOU P. Joint service caching and task offloading for mobile edge computing in dense networks[C]//IEEE INFOCOM 2018-IEEE Conference on Computer Communications. Piscataway: IEEE Press, 2018: 207-215.

[2] CHEN X, JIAO L, LI W Z, et al. Efficient multi-user computation offloading for mobile-edge cloud computing[J]. IEEE/ACM Transactions on Networking, 2016, 24(5): 2795-2808.

[3] DAI Y Y, XU D, MAHARJAN S, et al. Joint computation offloading and user association in multi-task mobile edge computing[J]. IEEE Transactions on Vehicular Technology, 2018, 67(12): 12313-12325.

[4] ZHOU Y Q, TIAN L, LIU L, et al. Fog computing enabled future mobile communication networks: a convergence of communication and computing[J]. IEEE Communications Magazine, 2019, 57(5): 20-27.

[5] QI Y L, TIAN L, ZHOU Y Q, et al. Mobile edge computing-assisted admission control in vehicular networks: the convergence of communication and computation[J]. IEEE Vehicular Technology Magazine, 2019, 14(1): 37-44.

[6] ZHOU Y Q, LIU L, WANG L, et al. Service-aware 6G: an intelligent and open network based on the convergence of communication, computing and caching[J]. Digital Communications and Networks, 2020, 6(3): 253-260.

[7] KRÓL M, MASTORAKIS S, ORAN D, et al. Compute first networking:

distributed computing meets ICN[C]//The 6th ACM Conference on Information-Centric Networking. New York: ACM Press, 2019: 67-77.

[8] 刘泽宁, 李凯, 吴连涛, 等. 多层次算力网络中代价感知任务调度算法[J]. 计算机研究与发展, 2020, 57(9): 1810-1822.
LIU Z N, LI K, WU L T, et al. CATS: cost aware task scheduling in multi-tier computing networks[J]. Journal of Computer Research and Development, 2020, 57(9): 1810-1822.

[9] PORAMBAGE P, OKWUIBE J, LIYANAGE M, et al. Survey on multi-access edge computing for Internet of things realization[J]. IEEE Communications Surveys & Tutorials, 2018, 20(4): 2961-2991.

[10] TALEB T, SAMDANIS K, MADA B, et al. On multi-access edge computing: a survey of the emerging 5G network edge cloud architecture and orchestration[J]. IEEE Communications Surveys & Tutorials, 2017, 19(3): 1657-1681.

[11] LEBRE A, PASTOR J, SIMONET A, et al. Revising OpenStack to operate fog/edge computing infrastructures[C]//2017 IEEE International Conference on Cloud Engineering (IC2E). Piscataway: IEEE Press, 2017: 138-148.

[12] NIU Y P, LIU F M, LI Z P. Load balancing across microservices[C]//IEEE INFOCOM 2018 - IEEE Conference on Computer Communications. Piscataway: IEEE Press, 2018: 198-206.

[13] 邬江兴. 新型网络技术发展思考[J]. 中国科学: 信息科学, 2018, 48(8): 1102-1111.
WU J X. Thoughts of the development of novel network technology[J]. SCIENTIA SINICA Informationis, 2018, 48(8): 1102-1111.

[14] 朱建军, 方琰崴. 面向服务的 5G 云原生核心网及关键技术研究[J]. 数字通信世界, 2018(2): 111.
ZHU J J, FANG Y W. Research on service oriented 5G cloud native core network and its key technologies[J]. Digital Communication World, 2018(2): 111.

[15] 王全. 云原生 Cloud Native 核心网方案及关键技术[J]. 中国新通信, 2018, 20(9): 61-62.
WANG Q. Cloud native core network scheme and key technologies[J]. China New Telecommunications, 2018, 20(9): 61-62.

[16] 胡聪丛. 无服务器计算的现状以及所面临的挑战[J]. 网络安全技术与应用, 2019(12): 84-85.
HU C C. The current situation and challenges of serverless computing[J]. Network Security Technology & Application, 2019(12): 84-85.

[17] HUNG Y H, WANG C Y. Fog micro service market: promoting fog computing using free market mechanism[C]//2018 IEEE Wireless Communications and Networking Conference (WCNC). Piscataway: IEEE Press, 2018: 1-6.

[18] YU R, KILARI V T, XUE G, et al. Load balancing for interdependent IoT microservices[C]//IEEE Conference on Computer Communications. Piscataway: IEEE Press, 2019.

[19] LI W B, LEMIEUX Y, GAO J, et al. Service mesh: challenges, state of the art, and future research opportunities[C]//2019 IEEE International Conference on Service-Oriented System Engineering (SOSE). Piscataway: IEEE Press, 2019: 122-1225.

[20] VILLARI M, CELESTI A, TRICOMI G, et al. Deployment orchestration of microservices with geographical constraints for edge computing[C]//2017 IEEE Symposium on Computers and Communications (ISCC). Piscataway: IEEE Press, 2017: 633-638.

[21] WANG Y, ZHAO C, YANG S, et al. MPCSM: microservice placement for edge-cloud collaborative smart manufacturing[J]. IEEE Transactions on Industrial Informatics, 2020.

[作者简介]

曾德泽（1984− ），男，中国地质大学（武汉）教授、博士生导师，主要研究方向为边缘计算、未来网络技术、物联网等。

陈律昊（1996− ），男，中国地质大学（武汉）硕士生，主要研究方向为边缘计算、微服务部署等。

顾琳（1985− ），女，华中科技大学副教授，主要研究方向为边缘计算、未来网络技术等。

李跃鹏（1994− ），男，中国地质大学（武汉）博士生，主要研究方向为边缘计算、任务调度、可信执行环境等。